Improved Learning in Convolutional Neural Networks with Shifted Exponential Linear Units (ShELUs)

Bertil Grelsson and Michael Felsberg

The self-archived postprint version of this conference article is available at Linköping University Institutional Repository (DiVA):

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-151606>

N.B.: When citing this work, cite the original publication.

Grelsson, B., Felsberg, M., (2018), Improved Learning in Convolutional Neural Networks with Shifted Exponential Linear Units (ShELUs), *2018 24th International Conference on Pattern Recognition (ICPR)*, , 517-522. <https://doi.org/10.1109/ICPR.2018.8545104>

Original publication available at:

<https://doi.org/10.1109/ICPR.2018.8545104>

Copyright: IEEE

<http://www.ieee.org/>

©2018 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Improved Learning in Convolutional Neural Networks with Shifted Exponential Linear Units (ShELUs)

Bertil Grelsson

Computer Vision Laboratory, Linköping University, Sweden
Email: bertil.grelsson@liu.se

Michael Felsberg

Computer Vision Laboratory, Linköping University, Sweden
Email: michael.felsberg@liu.se

Abstract—The Exponential Linear Unit (ELU) has been proven to speed up learning and improve the classification performance over activation functions such as ReLU and Leaky ReLU for convolutional neural networks. The reasons behind the improved behavior are that ELU reduces the bias shift, it saturates for large negative inputs and it is continuously differentiable. However, it remains open whether ELU has the optimal shape and we address the quest for a superior activation function.

We use a new formulation to tune a piecewise linear activation function during training, to investigate the above question, and learn the shape of the locally optimal activation function. With this tuned activation function, the classification performance is improved and the resulting, learned activation function shows to be ELU-shaped irrespective if it is initialized as a ReLU, LReLU or ELU. Interestingly, the learned activation function does not exactly pass through the origin indicating that a shifted ELU-shaped activation function is preferable. This observation leads us to introduce the Shifted Exponential Linear Unit (ShELU) as a new activation function.

Experiments on Cifar-100 show that the classification performance is further improved when using the ShELU activation function in comparison with ELU. The improvement is achieved when learning an individual bias shift for each neuron.

I. INTRODUCTION

The classification accuracy of Convolutional Neural Networks (CNNs) has improved remarkably over the last years. The reason for the improvement is manifold: more sophisticated layer designs [1], [2], effective regularization techniques reducing overfitting such as dropout [3] and batch normalization [4], new nonlinear activation functions [5], [6], improved weight initialization methods [2], [7], data augmentation and large scale data as ImageNet [8].

In this work, we focus on the nonlinear activation function and its effect on the network learning behavior. Since the introduction of the Rectified Linear Unit (ReLU) [9], it is generally accepted that the activation should be noncontractive to avoid the vanishing gradient problem. The vanishing gradient hampered the learning for the sigmoid and tanh activations.

Centering the activation, i.e. reducing the bias shift, is claimed to speed up learning [10]. When the Exponential Linear Unit (ELU) was introduced by Clevert et al. [5], one of the reason for its success and fast learning capability was claimed to be that the activation saturates for large negative inputs. ELU is also controlled by a hyperparameter that determines the saturation level. Another activation that is

Table I
ACTIVATION FUNCTIONS.

Activation	$x > 0$	$x \leq 0$
ReLU	x	0
LReLU	x	αx
SReLU	x	$\max(x, -1)$
ELU	x	$\alpha(\exp(x)-1)$
PELU	x	$\alpha(\exp(\beta x)-1)$

saturated for negative inputs is the Shifted ReLU (SReLU). Its shape is similar to ReLU, but the "kink" is at -1 instead of 0. This reduces the bias shift while being saturated for negative inputs. ELU learns both faster and better than SReLU [5], but it is not obvious which properties of ELU that actually create this improvement. It may be the smooth exponential decay for small negative inputs and/or the fact that it is continuously differentiable. The question also remains whether the shape of ELU is truly the optimal activation function or if there are other shapes, not yet found, that would further speed up and improve learning. And if they exist, how are they to be found. These were the type of issues that we wanted to explore when starting this work.

To improve the learning capabilities for the above mentioned activation functions, variants of them have been published where the control parameters are tuneable and learned instead of being set as a constant parameter according to their original publications. The Parametric ReLU (PReLU) was introduced by He et al. [2] where the single control parameter α for LReLU is now learned during training. The Parametric ELU (PELU) was introduced by Trottier et al. [6], also tuning the control parameters for ELU. Classification results were shown to improve with parameter tuning in both papers. The Scaled ELU (SELU)¹ was defined by Klambauer et al. [11] and is essentially a more simple variant of PELU. The activation functions mentioned so far are defined in Table I.

Piecewise linear activation functions have previously been used to improve the performance compared to LReLU [12]. In this work, we use the same concept with tuneable piecewise linear activation functions, but now with the additional objective to investigate the shape of the learned activation function.

¹SELU is a variant of PELU and to avoid confusion we name the Shifted ELU ShELU.

We apply this approach for nonlinear regression of the optimal activation function. We initialized the activation as linearized versions of ReLU, LReLU and ELU, and they *all* resulted in the same shape after tuning the network. The ReLU and LReLU activation functions are tuned into an ELU-shaped function whereas the ELU activation function retains its shape. However, we also noted that the tuned ELU-shaped activation function does not exactly pass through the origin. There is a small shift introduced around the origin while retaining the overall shape of the activation function.

Based on this observation, we introduce a shifted variant of the ELU activation function. In our experiments, we found that a *horizontal* shift is favorable and we call this new activation function Shifted Exponential Linear Unit (ShELU). The shift is tuneable during training and the shift is individual for each neuron. Experiments show that the classification performance is improved when allowing this shift in the activation function.

Our main contribution is the introduction of the shifted activation function ShELU. The second contribution is a new formulation of a tuneable piecewise linear activation function with constraints to make it continuous. This formulation can be used to explore for other, up to now unseen, shapes of activation functions. The third contribution is experimental support that an ELU-shaped activation function is favorable for learning; the tuneable piecewise linear activation function adapts to an ELU-shape during training, but with a small shift around the origin.

II. PIECEWISE LINEAR ACTIVATION FUNCTIONS

Piecewise linear activation functions were first introduced by Agostinelli et al. [12]. In this work, we use the same idea but now with the additional objective to investigate the shape of the learned activation function. If we initialize the activation function as a linearized version of ReLU, LReLU or ELU, how will the shape be changed during training?

Our formulation of a piecewise linear and continuous activation function consists of two steps: first a soft histogram is formed as in Felsberg and Granlund [13], second, a weighted sum of the histogram outputs is computed. The piecewise linear activation functions are learned individually for each neuron. The main difference in the formulations of the piecewise linear activation function is that Agostinelli et al. use a variable segment length of hinges where the learned parameters are the slopes and locations for the hinges. We use a constant segment length in our formulation where the learned parameters are the slopes and offsets for the segments.

A. Soft histogram

A soft histogram can be represented by two components; one offset component and one histogram component. We use N bins for positive input values and another N bins for negative input values. All bins in the histogram have constant and unity width. The bin limits take integer values and the bin centers are at -0.5, 0.5, 1.5 etc. The concept of the soft histogram is illustrated in Figure 1. In the figure, $N = 4$, but in the experiments we also used more bins like N equal to 8 and

16. Within a certain activation layer, we extract the maximum and the minimum input over a minibatch. We then linearly scale the positive input values to lie in the range $[0 N]$ and the negative input values to lie in the range $[-N 0]$. Now, as an example, consider two units with scaled input values of 2.68 and -1.80, respectively. For the first unit, the histogram component will be 1 for the bin centered at 2.5 and 0 for all other bins. The corresponding offset component will be 0.18, i.e. the signed distance from the bin center. For the second unit, the histogram component will be 1 for the bin centered at -1.5 and 0 for all other bins. The corresponding offset component will be -0.30. The output from the soft histogram for the two units will be

$$y(2.68) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0.18 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (1a)$$

$$y(-1.80) = \begin{bmatrix} 0 & 0 & -0.30 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1b)$$

An analytical formulation of the soft histogram can be given as

$$y(x) = \begin{bmatrix} (x - \text{floor}(x) - 0.5)m_\nu(x) \\ m_\nu(x) \end{bmatrix}_{2 \times 2N} \quad (2)$$

$$= \begin{bmatrix} o_0 & o_1 & \dots & o_{2N-1} \\ h_0 & h_1 & \dots & h_{2N-1} \end{bmatrix},$$

where $m_\nu(x)$ denotes the membership of the respective bins. The membership is 1 if the unit belongs to that bin and 0 for all other bins. The soft histogram output can alternatively be expressed with the offset and histogram components as in the right hand side of (2).

For backpropagation we need to compute the derivative of the soft histogram. In our formulation, we consider the membership $m_\nu(x)$ to be locally constant. This leads to a particular choice of subgradient that is also used in most implementations of max pooling, and it works well in practice. The derivative is computed as

$$\frac{dy}{dx} = \begin{bmatrix} m_\nu(x) \\ 0 \end{bmatrix}_{2 \times 2N} \quad (3)$$

The output from the soft histogram is independent of the activation function, but any activation function can be represented or approximated by a weighted sum of the histogram output.

B. Weighted sum

Different piecewise linear activation functions can now be realized by varying the weights applied to the soft histogram outputs. For each activation layer, we define a matrix W

Figure 1. Soft histogram decomposed into rectangular and linear basis functions.

Figure 2. Weighted sum examples, ReLU activation (top) and LReLU activation (bottom).

with weights for the offset components and the histogram components

$$W = \begin{bmatrix} w_{o_0} & w_{o_1} & \dots & w_{o_{2N-1}} \\ w_{h_0} & w_{h_1} & \dots & w_{h_{2N-1}} \end{bmatrix}. \quad (4)$$

To obtain a ReLU activation function, we set all weights to 1 for the offset components on the positive side and 0 for all offset components on the negative side. The weights for the offset components correspond to the slope of the activation function for each linear piece. Further, for a ReLU, we set the weights for the histogram components to 0.5, 1.5, 2.5, etc. on the positive side and to 0 on the negative side. The weights for the histogram components correspond to the bias level at the bin centers of the activation function for each linear piece.

To obtain a LReLU activation function, the weights (slopes) for the negative offset components are set to the value for the hyperparameter α , e.g. to 0.1. The weights for the offset components on the positive side are the same as for ReLU. For $\alpha = 0.1$, the weights for the histogram components on the negative side are set to -0.35, -0.25, -0.15 and -0.05, i.e. the bias level at the bin centers.

To summarize, the weight matrices for the ReLU and LReLU activation functions are defined as

$$W_{\text{ReLU}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0.5 & 1.5 & 2.5 & 3.5 \end{bmatrix} \quad (5a)$$

$$W_{\text{LReLU}} = \begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 1 & 1 & 1 & 1 \\ -0.35 & -0.25 & -0.15 & -0.05 & 0.5 & 1.5 & 2.5 & 3.5 \end{bmatrix}. \quad (5b)$$

The output for the weighted sum is the sum of an elementwise multiplication of the soft histogram with the weight matrix

$$y = \sum_{\nu} w_{o_{\nu}} o_{\nu} + w_{h_{\nu}} h_{\nu}. \quad (6)$$

The weighted sum output for the two example units is illustrated in Figure 2. For the ReLU activation, the outputs will be $2.5 \times 1 + 1 \times 0.18 = 2.68$ and $0 \times 1 + 0 \times (-0.30) = 0$, respectively. For the LReLU activation, the output for the unit with input value -1.80 will be $-0.15 \times 1 + 0.1 \times (-0.30) = -0.18$, as desired.

Our formulation will generate a piecewise linear activation function for any values chosen as weights for the offset and

Table II
TOP1 ERROR ON CIFAR-100 WITH LENET NETWORK.

Activation	Top1 error (%)
ReLU	46.58
Tuned ReLU	45.92
LReLU	45.41
Tuned LReLU	45.18
ELU	44.96
Tuned ELU	44.51

histogram components. However, it is obvious that constraints need to be put on the weights if a *continuous* activation function is to be obtained. The weights (slopes) for the offset components can be set independently but only one weight for the histogram component is independent from the other weights. Assume that w_{h_0} is set as desired. To obtain a continuous linear function, the remaining histogram weights then need to be set as

$$\begin{aligned} w_{h_1} &= w_{h_0} + 0.5(w_{o_0} + w_{o_1}) \\ w_{h_2} &= w_{h_0} + 0.5(w_{o_0} + 2w_{o_1} + w_{o_2}) \\ &\vdots \\ w_{h_{2N-1}} &= w_{h_0} + 0.5(w_{o_0} + 2w_{o_1} + \dots + 2w_{o_{2N-2}} + w_{o_{2N-1}}). \end{aligned} \quad (7)$$

The constraints in (7) must be enforced when updating the weights in the backpropagation step.

C. Evaluation of piecewise linear activation functions

To investigate the behavior of the piecewise linear activation function we made experiments with the Lenet network [14] and the Cifar-100 dataset [15]. We used the implementation of Lenet as provided when downloading the MatConvNet [16] framework. We ran the Lenet network with the ReLU activation function, and also replaced all activation layers with LReLU and ELU. We then exchanged the activation layers with the piecewise linear activation layer. We initialized the layers as a linear version of ReLU, LReLU and ELU respectively. We consistently noticed a slight improvement (a few tenths of a percent) in classification performance when using the tuneable piecewise linear activation function compared to its corresponding fixed activation function, see Table II. Besides the slight classification improvement, it is also interesting to analyse the shape of the activation functions after tuning, see Figure 3. All three activation functions remain linear and with almost unity slope on the positive side. All three tuned activation functions exhibit a smooth “exponential” decay for small negative inputs and then remain fairly constant for larger negative inputs. The resulting shape after tuning for all three initializations is close to the ELU shape. However, notice that all tuned activation functions tend to return a variable but negative output for zero input and that they do not pass through the origin. These results suggest that we introduce the Shifted Exponential Linear Unit (ShELU) as an activation function. From the results it is not obvious whether the introduced shift around the origin should be vertical or horizontal. For a horizontal shift, the saturation level remains constant for large negative inputs which may seem more

Figure 3. Initialization (red) and 20, 50 and 80 percentiles for tuned activation functions in last layer; ReLU (left), LReLU (middle) and ELU (right).

intuitive. For a vertical shift, the saturation level will vary depending on the shift which better matches the achieved results on the Lenet network.

III. SHIFTED ACTIVATION FUNCTIONS

The results presented in section II-C show that an ELU-shaped activation function which is shifted around the origin seems favorable to improve learning. Hence, we introduce the ShELU activation function with *horizontal* shift and the SvELU activation function with *vertical* shift and define them as in Table III. The hyperparameter α is considered to be a pre-set constant and it is not tuned during training. In our experiments, we set $\alpha = 1$. However, we also define PShELU, a variant of PELU with horizontal shift. The parameters α and β for PShELU are learned during training. In the experiments, they were initialized as $\alpha = \beta = 1.0$, i.e. as an original ELU activation.

Note that the introduced shifts δ in Table III are individual for all neurons. As an example, consider the first layers in the Lenet network shown in Figure 4. The input is an image $32 \times 32 \times 3$. In the first convolutional layer, there are 192 filters with size $5 \times 5 \times 3$. The output from the convolutional layer consists of 192 feature maps with size 32×32 . The output includes a bias level for each feature map (each large square in the convolutional output), i.e. a total of 192 bias levels. The activation function is applied to the individual neurons resulting in an output with size $32 \times 32 \times 192$. When we say that we introduce individual shifts for all neurons, it means that there is one tuneable shift for each of the $32 \times 32 \times 192$ neurons (all small squares in the activation output) where the activation function is applied.

In Goodfellow et al. [17] (chapter 9.5), it is stated that for CNNs it is natural to have shared biases with the same tiling pattern as the convolutional kernels, but that individual biases for each neuron “would allow the model to correct for differences in the image statistics at different locations”.

Table III
SHIFTED ACTIVATION FUNCTIONS.

Activation	Value	Region 1	Value	Region 2
ShELU	$x + \delta$	$x + \delta > 0$	$\alpha(\exp(x + \delta) - 1)$	$x + \delta \leq 0$
SvELU	$x + \delta$	$x > 0$	$\alpha(\exp(x) - 1) + \delta$	$x \leq 0$
PShELU	$\frac{\alpha}{\beta}(x + \delta)$	$x + \delta > 0$	$\alpha(\exp(\frac{x + \delta}{\beta}) - 1)$	$x + \delta \leq 0$

Figure 4. Bias levels for convolutional layers and shifts for activation function. Biases/shifts can either be shared (large squares) or individual for each neuron (small squares).

By introducing the activation function ShELU, with individual shifts for each neuron, we have indirectly created individual biases for the convolutional layer feature map output. Note that a convolutional layer with a shared bias level for each feature map output followed by a ShELU activation is equivalent with a convolutional layer with individual bias levels for each feature map output followed by an ELU activation. This equivalence was verified with experiments presented in section IV-A1. However, frameworks as Caffe [18] and MatConvNet [16] do not allow for individual biases in a convolutional layer but is restricted to shared biases.

IV. EXPERIMENTS

A. Experiments with shifted activation functions

1) *Experiments on Cifar-100 with Lenet network:* We now want to evaluate if the classification performance improves with the new activation functions ShELU and SvELU compared to ELU. We start with the Lenet network and replace all ELU activations with either the ShELU or the SvELU activation. Image data was preprocessed with global contrast normalization and whitening [19]. Note that the complete dataset was divided by a factor of 10 (compared to the preprocessing provided with the MatConvNet download) to better match the variance with Xavier initialization. During training the dataset was augmented with random horizontal flipping and by randomly cropping images from the original images zero padded with a frame of width four.

The toplearnings in Table IV are the average over 8 runs for each activation function. The results show that there is a small improvement on the toplearning using the ShELU and SvELU activation functions compared with the original ELU. Furthermore, the shifted activation function PShELU achieves a

Table IV
TOP1 TEST ERRORS ON CIFAR-100 WITH LENET, CLEVERT-11 AND CLEVERT-18 NETWORKS.

Activation	Lenet	Clevert-11	Clevert-18
ELU	44.96	28.76	25.16
ShELU	44.77	28.57	25.03
SvELU	44.70	28.85	-
PELU	45.03	28.78	-
PShELU	44.76	28.74	-
ConvIndBias + ELU	44.78	-	-
ReLU	46.58	31.86	28.63

slightly better test results than both ELU and PELU. However, the significance of these results is limited as Lenet is a rather shallow network.

We also created a network layer named "ConvIndBias", which is an identity mapping but it also adds an individually learned bias shift for each neuron. The results in Table IV confirm that a ShELU activation is equivalent to the combination of a ConvIndBias layer and an ELU activation as was stated in section III.

2) *Experiments on Cifar-100 with Clevert-11*: To further evaluate the shifted activation functions in comparison with ELU, we built the 11-layer network designed by Clevert et al. [5] to replicate the experiments when ELU was introduced. We denote the network Clevert-11. Parameter settings and weight initializations were as in [5]. Our classification results with the network Clevert-11 on the Cifar-100 dataset for the activation functions ELU, ShELU, SvELU, PELU, PShELU and ReLU are presented in Figure 5 and summarized in Table IV. Our results are the average over 9 runs for each activation function. The results in the table are the average top1error over the last 20 epochs for each activation function.

The results show that the test error for the ShELU activation function is significantly better than for ELU, whereas the error for SvELU is slightly inferior. The results suggest that a *horizontal* shift for the activation function is preferable to a *vertical* shift. The training behavior is almost identical for ELU and ShELU. We believe that the improved test result can be attributed to that ShELU adaptively learns where to set the reference level between the linear and exponential parts of the activation function.

ELU, PELU and PShELU all show very similar test errors. Note, however, that the training error is by far lower for PELU indicating pronounced overfitting compared to ELU. The training error is lower for PShELU than for ShELU but the test error is inferior. This suggests that PShELU suffers from overfitting when allowed to tune the hyperparameters α and β . Note that we were able to almost exactly reproduce the results for ELU achieved in [5] who report a top1error of 28.75%. The train and test errors for ReLU are significantly higher than for all ELU-shaped activation functions.

3) *Experiments on Cifar-100 with Clevert-18*: The last experiments where ShELU is evaluated are performed with the 18-layer network designed by Clevert et al. [5]. As described in [5], we introduce dropout layers after *all* layers in a stack as a second training phase, and then increase the dropout rate

Figure 5. Left column: Training (dashed) and test (solid) errors on Cifar-100 with network Clevert-11 (top). Test errors (final part) for ELU, ShELU and SvELU (middle), and ELU, PELU and PShELU (bottom). Right column: Training (dashed) and test (solid) errors on Cifar-100 with network Clevert-18 (top). Training errors (middle) and test errors (bottom) for final part.

for fine tuning in a third phase. We modified the learning rate scheme to obtain as low average test errors as we possibly could for ELU and used the same scheme for ShELU. We use the learning rate 0.01 for the first 100 epochs and then decrease by a factor of 10 for another 40 epochs. We train for 60 epochs in the second phase and lower the learning rate to 0.0001 entering the third phase. After 60 epochs, we decrease the learning rate by a factor of 10 for the last 40 epochs.

The results shown in Figure 5 are the average over five runs and the errors in Table IV are the average over the last 20 epochs. The test error for ShELU 25.03% is significantly better than for ELU, even for this network which produces state-of-the-art results. The training error is lower for ShELU than for ELU during fine tuning, which indicates that the individual biases improve the learning capability for this network. The test error achieved with ELU in [5] is 24.28%, but it is unclear whether this result is obtained as an average or as a best run.

The test error for ReLU is significantly higher than for ShELU and ELU, and the difference in test error increases with the depth of the network.

B. Learned shifts for ShELU

In all experiments, we initialized the individual shifts for the ShELU activation from a Gaussian distribution with standard deviation 0.001. The learned shifts after training in the 10 activation layers of the Clevert-11 network are shown as

Figure 6. Learned shifts for ShELU activation function, relative frequency (top), kurtosis (lower left) and spatial variation (lower right).

normalized frequency histograms in Figure 6, together with the kurtosis and the spatial variation for the shifts. The shape of the learned shifts is almost a perfect Gaussian distribution for all layers. This is supported by the computed kurtosis which is close to 3.0. The kurtosis increases slightly for the last three layers where the distribution tends to be somewhat skewed towards the negative side. The standard deviation for the shift is relatively constant for the first nine layers but grows considerably for the last layer. Figure 6 shows the learned shifts for the first ShELU activation layer where the shifts for the 192 feature maps have been placed as 12×16 tiles side by side. Interestingly, the spatial variation for the learned shift seems to be completely random. Any statistical difference spatially over the image cannot be perceived.

V. CONCLUSION

We use a new formulation to tune a continuous piecewise linear activation function during training and learn the shape of the locally optimal activation function. With this tuned activation function, the classification performance for convolutional neural networks is improved and the resulting, learned activation function shows to be ELU-shaped irrespective whether it is initialized as a ReLU, LReLU or ELU activation function. The learned activation function exhibits a variable shift around the origin for each neuron, indicating that a shifted ELU-shaped activation function is preferable. This observation leads us to introduce the Shifted Exponential Linear Unit (ShELU) as a new activation function.

Experiments on Cifar-100 show that the classification performance is further improved when using the ShELU activation function in comparison with ELU. Normally in a convolutional network layer, one shared bias shift is learned for each feature map output. The improvement for the ShELU activation is achieved when learning an individual bias shift for each neuron. The equivalent to the ShELU activation function would be to learn an individual bias shift for each neuron

in the convolutional layer output and then apply an ELU activation, which however is not supported by commonly used deep learning frameworks. The implementation of individual biases in the activation function is therefore preferable and leads to the ShELU activation function.

ACKNOWLEDGMENT

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

This research is funded by the The Swedish Research Council through a framework grant for the project *Energy Minimization for Computational Cameras* (2014-6227).

REFERENCES

- [1] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.
- [2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [3] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456.
- [5] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.
- [6] L. Trottier, P. Giguère, and B. Chaib-draa, "Parametric exponential linear unit for deep convolutional neural networks," *arXiv preprint arXiv:1605.09332*, 2016.
- [7] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.
- [8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.
- [9] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.
- [10] Y. Le Cun, I. Kanter, and S. A. Solla, "Eigenvalues of covariance matrices: Application to neural-network learning," *Physical Review Letters*, vol. 66, no. 18, p. 2396, 1991.
- [11] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *arXiv preprint arXiv:1706.02515*, 2017.
- [12] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," *arXiv preprint arXiv:1412.6830*, 2014.
- [13] M. Felsberg and G. Granlund, "P-channels: Robust multivariate estimation of large datasets," in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 3. IEEE, 2006, pp. 262–267.
- [14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [15] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.
- [16] <http://www.vlfeat.org/matconvnet/v.beta-20>.
- [17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.
- [19] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.